01 Jan 2005

Negative Reinforcement and Backtrack-Points for Recurrent Neural Networks for Cost-Based Abduction

Donald C. Wunsch
Missouri University of Science and Technology, dwunsch@mst.edu

Ashraf M. Abdelbar

M. A. El-Hemaly

Emad A. M. Andrews

Negative Reinforcement and Backtrack-Points for Recurrent Neural Networks for Cost-Based Abduction

Ashraf M. Abdelbar, Mostafa A. El-Hemaly, Emad A.M. Andrews
Department of Computer Science, American University in Cairo

Donald C. Wunsch II
University of Missouri, Rolla

Abstract— **Abduction is the process of proceeding from data describing a set of observations or events, to a set of hypotheses which best explains or accounts for the data. Cost-based abduction (CBA) is an AI formalism in which evidence to be explained is treated as a goal to be proven, proofs have costs based on how much needs to be assumed to complete the proof, and the set of assumptions needed to complete the least-cost proof are taken as the best explanation for the given evidence. In this paper, we introduce two techniques for improving the performance of high order recurrent networks (HORN) applied to cost-based abduction. In the backtrack-points technique, we use heuristics to recognize early that the network trajectory is moving in the wrong direction; we then restore the network state to a previously-stored point, and apply heuristic perturbations to nudge the network trajectory in a different direction. In the negative reinforcement technique, we add hyperedges to the network to reduce the attractiveness of local-minima. We apply these techniques on a 300-hypothesis, 900-rule particularly-difficult instance of CBA.**

I. Introduction

Cost-based abduction (CBA) is an important problem in diagnostic reasoning under conditions of uncertainty [10]. In CBA, evidence to be explained is treated as a goal to be proven, proofs have costs based on how much needs to be assumed to complete the proof, and the set of assumptions needed to complete the least-cost proof (LCP) are taken as the best explanation for the given evidence. In previous work [5], we applied high-order recurrent networks (HORN) to CBA. In this paper, we apply HORN's to a large, particularly-difficult CBA instance containing 300 hypotheses and 900 rules. We present two novel techniques for improving the quality of solution returned by the HORN: backtrack-points (Sect. V) and negative reinforcement (Sect. VI).

We begin, in Section II, with a review of HORN's and penalty logic (PL). Section III defines CBA and surveys previous work. Section IV presents the basic algorithm for transforming a CBA instance into a HORN. Sections V and VI present the backtrack-points and negative reinforcement techniques, respectively, along with experimental results. Section VII presents some discussion of efficient implementation and parallelization, and Section VIII concludes with final remarks and future outlook.

II. Penalty Logic & High Order Networks

A. High Order Recurrent Networks

A recurrent neural network is one whose underlying topology of inter-neuronal connections contains at least one cycle. In the case of the Hopfield network [14], the underlying topology is a graph: each weighted connection is either an edge connecting two neurons, or a self-loop involving one neuron. A High Order Recurrent Network (HORN) [2], [27] is a recurrent network whose underlying topology is a hypergraph, *i.e.*, it allows weighted hyperedges which connect more than two neurons. The degree of a hyperedge is the number of neurons it connects; the order of a HORN is the largest hyperedge degree in the topology. We will use the notation $T^{(k)}_{i_1 \ldots i_k}$ to denote the weight of the k^{th}-degree edge connecting neurons $i_1 \ldots i_k$.

The input u_i to a neuron in a k^{th}-order HORN, whose output is V_i, can be characterized by

$$\Delta u_i = \sum_{\ell=1}^{k} \sum_{i_1 \ldots i \ldots i_\ell} T^{\ell}_{i_1 \ldots i \ldots i_\ell} \prod_{1 \le j \le \ell, i_j \neq i} V_{i_j} \ . \qquad (1)$$

A k^{th}-order HORN can be viewed as minimizing a k^{th}-order energy function [23], [27]:

$$\begin{aligned} E \ = \ & - \sum_{1 \le i_1 < i_2 < \ldots < i_k \le n} T^{(k)}_{i_1 \ldots i_k} V_{i_1} \ldots V_{i_k} \\ & - \sum_{1 \le i_1 < i_2 < \ldots < i_{k-1} \le n} T^{(k-1)}_{i_1 \ldots i_{k-1}} V_{i_1} \ldots V_{i_{k-1}} \\ & - \ldots - \sum_{1 \le i \le n} T^{(1)} V_i \ . \end{aligned} \qquad (2)$$

For example, the energy function of a fourth-order HORN has the form:

$$\begin{aligned} E \ = \ & - \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{k=j+1}^{n} \sum_{l=k+1}^{n} T^{(4)}_{ijkl} V_i V_j V_k V_l \\ & - \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{k=j+1}^{n} T^{(3)}_{ijk} V_i V_j V_k \\ & - \sum_{i=1}^{n} \sum_{j=i+1}^{n} T^{(2)}_{ij} V_i V_j - \sum_{i=1}^{n} T^{(1)}_{i} V_i \ . \end{aligned} \qquad (3)$$

0-7803-9048-2/05/$20.00 ©2005 IEEE

The relationship between the output V_i and the activation level u_i of a neuron i can follow the form either of the Hopfield network [14], of the stochastic Boltzmann machine [12], or of the mean field annealing Boltzmann machine [22]. In our implementation, we use the mean field annealing Boltzmann machine: for a given neuron i,

$$V_i = \text{sigmoid}(u_i/t) \ , \tag{4}$$

where t is a temperature parameter that starts at an initial value t_0, and is decreased by a fixed percentage factor t_f every e_t epochs (where an epoch is a full pass through all the neurons).

B. Penalty Logic

Penalty logic (PL) is an extension of propositional logic which associates real-valued penalties with wff's (well-formed formulas) and allows the wff's to be violated at a cost of their associated penalties. A PL $\mathcal{L}$ is a set of pairs $\{\langle \rho_i, \phi_i \rangle\}$, where ϕ_i is a propositional-logic wff called an *assumption* and ρ_i is a real-valued penalty. A truth assignment to the set of propositions which participate in the assumptions of $\mathcal{L}$ is called a *world model*. If a given assumption does not have a value of true under a given world model, then it is said to be *violated*. The *violation rank* of a given world model is the sum of the penalties associated with all the assumptions that are violated by the model.

Consider the following example $\mathcal{L}$ over the set of propositions $\{p, q, r, s\}$.

100	$p\overline{r} \rightarrow s$	75	$q\overline{r} \vee \overline{q}r$	500	p
200	$s \leftrightarrow \overline{q}$	50	$\overline{r} \vee \overline{p}$	120	q

Here, the minimum-violation rank world model is $\{p \leftarrow \mathbf{T}, r \leftarrow \mathbf{F}, s \leftarrow \mathbf{F}, q \leftarrow \mathbf{T}\}$ with penalty equal to 100.

C. Equivalence between Penalty Logic and HORN's

Pinkas [23] shows that there is an equivalence between HORN's and penalty logic, specifically between the problem of finding the minimum energy neuron assignment for a HORN and the problem of finding the minimum violation rank world model for a PL. We briefly sketch the basic elements of this transformation here, but the reader is referred to [23] for a fuller description.

Given a logical expression s, a characteristic function $H(s)$ is constructed to have its maximum value exactly when s has a value of true. If $s = s_1 \wedge s_2$, then

$$H(s) = H(s_1) \times H(s_2) \ . \tag{5}$$

If $s = s_1 \vee s_2$, then

$$H(s) = H(s_1) + H(s_2) - H(s_1) \times H(s_2) \ . \tag{6}$$

If $s = \neg s_1$, then

$$H(s) = 1 - H(s_1) \ . \tag{7}$$

Finally, if $s = x_1$, where x_1 is an atomic proposition, then

$$H(s) = x_1 \ , \tag{8}$$

where x_1 has values of 1 and 0 in the algebraic function if and only if it has values of true and false, respectively, in the logical function. For example, the expression $s = a \wedge b \rightarrow c$ would be converted into a characteristic function as follows:

$$\begin{aligned} H(a \wedge b \rightarrow c) &= H(\neg(a \wedge b) \vee c) = H((\neg a \vee \neg b) \vee c) \\ &= H(\neg a \vee \neg b) + c - H(\neg a \vee \neg b)c \\ &= (1-a) + (1-b) - (1-a)(1-b) + c \\ &\quad - c(1-a) - c(1-b) + c(1-a)(1-b) \\ &= 1 - ab + abc \ . \end{aligned} \tag{9}$$

We can see from the following truth table that $H(s)$ indeed has its maximal points exactly at those truth assignments which satisfy s.

a	b	c	s	H(s)	a	b	c	s	H(s)
0	0	0	T	1	0	0	1	T	1
0	1	0	T	1	0	1	1	T	1
1	0	0	T	1	1	0	1	T	1
1	1	0	F	0	1	1	1	T	1

For a given PL $\mathcal{L}$, the Pinkas transformation constructs a HORN as follows:

1) Each proposition in $\mathcal{L}$ becomes a neuron in the HORN.
2) For each $\langle \rho_i, \phi_i \rangle \in \mathcal{L}$, construct the characteristic function $H(\neg\phi_i)$ and add to the HORN the connections corresponding to $[\rho_i \times H(\neg\phi_i)]$.

III. Cost-Based Abduction

A. Introduction

Abduction is the process of proceeding from data describing observations or events, to a set of hypotheses, which best explains or accounts for the data [13]. A CBA system is a knowledge representation in which a given world situation is modeled as a 4-tuple $\mathcal{K} = (\mathcal{H}, \mathcal{R}, c, \mathcal{G})$, where

- $\mathcal{H}$ is a set of *hypotheses* or *propositions*,
- $\mathcal{R}$ is a set of *rules* of the form

  $$(h_{i_1} \wedge h_{i_2} \wedge \ldots \wedge h_{i_n}) \longrightarrow h_{i_k} \ ,$$

  where $h_{i_1}, \ldots, h_{i_n}$ (called the *antecedents*) and h_{i_k} (called the *consequent*) are all members of $\mathcal{H}$, and where $\mathcal{R}$ does not contain any *logical cyclicities* (see definition below),
- $\mathcal{G} \subseteq \mathcal{H}$ is called the *goal set* or the *evidence*.

If a hypothesis h_i is an antecedent for a rule for which hypothesis h_j is a consequent, then we say that h_i is an *ancestor* of h_j; further, if h_i is an ancestor of h_j, and h_j is an ancestor of h_k, then h_i is an ancestor of h_k. A *logical cyclicity* occurs if there are two hypothesis h and h' such that h is an ancestor of h', and h' is an ancestor of h.

The objective is to find the LCP for the evidence, where the cost of a proof is taken to be the sum of the costs of all hypotheses that must be assumed in order to complete the proof. Any given hypothesis can be made true in two ways: it can be assumed to be true, at a cost of its assumability cost, or it can be proved. If a hypothesis occurs as the consequent of a

rule R, then it can be proved, at no cost, to be true by making all the antecedents of R true, either by assumption or by proof. If a hypothesis does not appear as the consequent of any rule, then it cannot be proved, it can be made true only by being assumed. The cost of a hypothesis can be ∞, which means that it cannot be assumed, it can only be proved. One can assume, without loss of generality, that any hypothesis that appears as the consequent of any rule has an infinite assumability cost. We therefore consider the hypothesis set $\mathcal{H}$ to be partitioned into two subsets: a set of assumable hypotheses $\mathcal{H}^A$, which have finite assumability costs and do not appear as consequents of any rules, and a set of provable hypotheses $\mathcal{H}^P$, which have infinite assumability costs and, hence, can be made true only by being proved.

B. Literature Review

Finding an LCP for an instance of CBA was shown to be $\mathcal{NP}$-hard in 1994 [10], and, in 2004, Abdelbar [3] showed that even approximating an LCP within a fixed ratio bound of the optimal is $\mathcal{NP}$-hard.

A number of approaches to this problem have been explored. Charniak and Shimony [9], [10] presented a best-first heuristic search method, and admissible heuristics have been investigated by Charniak and Husain [8], and Abdelbar and Hefny [7].

Santos [24], [25] presented a method for transforming a CBA instance into a set of linear constraints, which could then be solved by 0-1 integer linear programming (ILP). Santos' operations research (OR) based approach was followed by several others: Ishizuka and Matsuo [15] presented a method called *slide down and shift up*, which uses a combination of linear programming and nonlinear programming to find approximate solutions in polynomial-time; Ohsawa and Ishizuka [20] presented a method called *bubble propagation*, which also finds approximate solutions in polynomial-time; Matsuo and Ishizuka [19] investigated linear and nonlinear programming approaches to CBA and to more general logical reasoning problems such as satisfiability. Santos and Santos [26] presented sufficient conditions for a CBA instance to be polynomially-solvable based on the idea of totally unimodular matrices; their work has been extended by Ohsawa and Yachida [21].

Abdelbar [1] showed that methods for cost-based abduction can be used for belief revision on belief networks. Kato *et al.* [16] investigated a method for finding LCP's based on binary decision diagrams. Den [11] presented a chart-based method for cost-based abduction. Kato *et al.* [17] investigated a search control mechanism for the A^* algorithm for cost-based abduction, and Kato *et al.* [18] investigated the parallelization of cost-based abduction with parallel best-first search. Recently, ant colony [6], and population-oriented simulated annealing [4] approaches to cost-based abduction have also been explored.

C. Generating CBA Instances

We used a CBA random instance generator, previously described in [5], which takes as parameters the total number of hypotheses, the number of rules, and a lower bound on the number of assumable hypotheses. Initial experimentation suggested a ratio of 1:3 for the total number of hypotheses relative to the number of rules yielded difficult instances. We then fixed the total number of hypotheses at 300, and the number of rules at 900, and allowed the number of assumable hypotheses to vary from 40 to 200, in steps of 10, generating 25 random instances at each step. Thus, the total number of CBA instances generated was 425 (these instances are available from **www.cbalib.org**). Out of these instances, a method was needed to choose the most likely to be difficult without exactly solving each instance. Each instance was converted to an integer linear program (ILP) using Santos' method [25]. The linear program (LP) relaxation of the ILP corresponding to each instance was solved using the popular public-domain engine `lp-solve`. The number of non-integral variables in the solution to the linear program, denoted f, was determined. The worst-case run-time of the full ILP is bounded from above by 2^f, although, of course, because of the branch-and-bound pruning process, the actual run-time may be much less than this. However, lacking another measure, we used the ratio of f to the total number of variables as a rough predictor of the difficulty of the problem instance. Based on this ratio, we selected for this paper the problem instance with the highest value for this ratio and used ILP (again using `lp-solve` as the engine) to obtain the exact solution for this instance. The characteristics of this instance and of the ILP solution are as follows:

- Instance name: `raa180` (available at **www.cbalib.org**).
- Number of hypotheses: 300; number of rules: 900.
- Assumable hypotheses: 180.
- Maximum rule depth: 38; average: 25.0; median: 27.
- Maximum number of rules in which a single hypothesis appears as a consequent: 15; average: 7.5; median: 7.
- Maximum number of rules in which a single hypothesis appears as an antecedent: 72; average: 14.6; median: 11.
- Number of global optimums: one unique optimal solution of cost 10,821.
- ILP CPU time: 88,835 seconds (24.68 hours).
- ILP branch-and-bound tree depth: 41.
- ILP tree nodes: 178,313.

IV. Basic Algorithm

In this section, we present an algorithm for converting a given instance $(\mathcal{H}, \mathcal{R}, c, \mathcal{G})$ of cost-based abduction into an instance $\mathcal{L} = \{\langle \rho_i, \phi_i \rangle\}$ of penalty logic, such that the least-cost proof for $(\mathcal{H}, \mathcal{R}, c, \mathcal{G})$ will correspond to the minimum violation-rank world model for $\mathcal{L}$.

Given a rule $R \in \mathcal{R}$, we will let $\alpha(R)$ denote the antecedent set (left-hand side) of R and $\beta(R)$ denote the consequent (right-hand side) of R. We will assume, without loss of generality, that $|\mathcal{G}| = 1$. Let $\Gamma_1 > 1$ and $\Gamma_2 > 1$ be two user-set constants.

Let c_∞ be equal to

$$c_\infty = 1 + \sum_{h \in \mathcal{H}^A} c(h). \quad (10)$$

For each $h \in \mathcal{H}^P$, define $\mathcal{R}_h = \{R \in \mathcal{R} | h = \beta(R)\}$. For each $R_i \in \mathcal{R}_h$, define

$$\phi_{hi} = (h_{j_1} \wedge h_{j_2} \wedge \ldots \wedge h_{j_n}) \ , \quad (11)$$

where $n = |\alpha(R_i)|$ and $\alpha(R_i) = \{h_{j_1}, h_{j_2}, \ldots, h_{j_n}\}$. Add to $\mathcal{L}$ the clause $\theta_h = \langle \rho_h, \phi_h \rangle$, where ρ_h equals $\Gamma_1 c_\infty$, and

$$\phi_h = \neg h \vee \phi_{h1} \vee \phi_{h2} \vee \ldots \vee \phi_{hk} \ , \quad (12)$$

where $k = |\mathcal{R}_h|$.

For each $h \in \mathcal{H}^A$, add to $\mathcal{L}$ the clause $\theta_h = \langle \rho_h, \phi_h \rangle$, where ϕ_h equals $\neg h$, and ρ_h equals $c(h)$.

Let the goal set consist of the hypothesis h. Add to $\mathcal{L}$ the clause $\theta_\mathcal{G} = \langle \rho_h, \phi_h \rangle$, where $\phi_h = h$, and $\rho_h = \Gamma_2 c_\infty$.

The minimum violation-rank world models of $\mathcal{L}$ will correspond exactly to the least-cost proofs of $(\mathcal{H}, \mathcal{R}, c, \mathcal{G})$. For each clause $\langle \rho_h, \phi_h \rangle \in \mathcal{L}$, Equations (5)-(8) are then used to obtain $H(\neg\phi_h)$. The algebraic expression $H(\neg\phi_h)$ will have its minimum point exactly when ϕ_h is true. If the algebraic function $H(\neg\phi_h)$ contains non-unity powers (i.e. x^k for some $k > 1$) of any of the hypotheses x, then x^k is substituted by x; the resulting function is denoted $H'(\neg\phi_h)$. The two functions $H(\neg\phi_h)$ and $H'(\neg\phi_h)$ will have the same global minima. Each algebraic function $H'(\neg\phi_h)$ is then multiplied by the weight ρ_h, and is expressed as a set of weighted hyperedges according to Equation (2).

V. Backtrack Points

A. Straightforward Application

Given a CBA instance, we can apply the algorithm of the previous section to obtain an equivalent PL instance and then apply the Pinkas transformation [23] (see Sect. II.C) to obtain an equivalent HORN. We will call this approach the *vanilla* approach.

The network obtained when the vanilla approach is applied to `raa180` has the following characteristics:

- Total neurons: 300.
- Total hyperedges: 95,348.
- Average number of hyperedges in which a single neuron participates: 8,231.1.
- Maximum number of hyperedges in which a single neuron participates: 37,953.
- Average number of neurons per hyperedge: 25.9.
- Maximum number of neurons per hyperedge: 55.

Because of the complexity of this HORN, it is not easy to obtain good, or even feasible, solutions using the vanilla approach. Even with high values for Γ_1 and Γ_2, which proved sufficient on most problems for producing feasible solutions, and with a variety of temperature cooling schedules, the network invariably falls into local minima that correspond to invalid solutions of the following kinds:

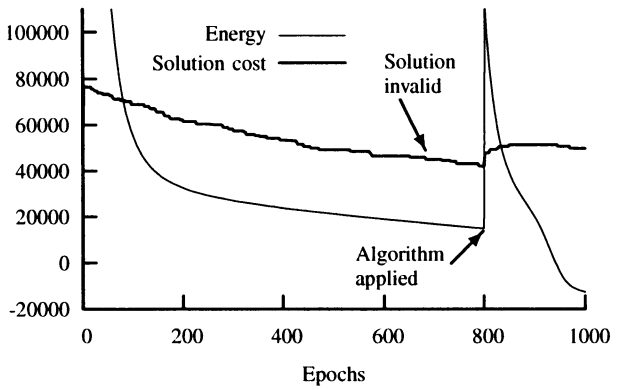

Fig. 1. Application of the backtrack-points to `raa180` (only showing the first 1000 epochs).

- The goal is assumed without proper proof (happens when $\Gamma_1 < \Gamma_2$).
- The goal is not reached (happens when $\Gamma_1 > \Gamma_2$).

B. Backtrack points

In order to avoid infeasible solutions in this (and other) problems, we developed a generic technique that exploits the underlying logic of the problem to successfully guide the network if it starts to fall into an invalid solution.

The motivation for this technique came from observing the behavior of the network while solving this and other large instances of CBA, especially when it falls into local minima corresponding to invalid solutions (these observations apply mainly to the case when $\Gamma_1 < \Gamma_2$). In these scenarios, the network would initially find a valid solution with a relatively high cost (a lot of hypotheses are assumed), and then as it is reducing the cost there would come a point when some of the hypotheses are deactivated even though they are really needed to prove the goal — resulting in an invalid solution. Another interesting observation is that in this initial phase it is almost always the case that not all the logical rules are utilized by the network: for some rules in the CBA, the neurons corresponding to all the antecedents are activated while the neuron corresponding to the consequent is deactivated.

To avoid this scenario, we started storing what we call *backtrack-points*. Every 200 epochs, as long as the network is in a state corresponding to a valid solution, we store the network state in a temporary buffer. Then, as soon as the network enters into a state corresponding to an invalid solution, we restore the network to the latest backtrack-point and apply the following perturbation algorithm:

Algorithm 1

1) Construct the set $\mathcal{T} \subset \mathcal{H}^A$ that contains all the hypotheses that correspond to neurons whose state in the network is > 0.5.
2) Assuming all the hypotheses in $\mathcal{T}$ to be true, repeatedly apply the rules in $\mathcal{R}$ in order to construct the set $\mathcal{Y} \subset$

$\mathcal{H}^{\mathcal{P}}$ that contains all the hypotheses that can be proved from $\mathcal{T}$.

3) For every neuron x in the network that corresponds to a hypothesis $h \in \mathcal{Y}$, if $V_x < 0.5$, adjust u_x so that you have $V_x > 0.5$.

Applying this stategy to `raa180`, we were able to achieve two important benefits:

1) Now the network consistently found valid solutions in every run.
2) Due to the robustness introduced by this strategy, we were able to employ low values for Γ_1 and Γ_2 (namely, $\Gamma_1 = 0.4$ and $\Gamma_2 = 0.8$), allowing the network to focus more on reducing the cost and thus obtaining better solutions.

Fig. 1 shows a typical run, with an initial temperature of $t_0 = 10^6$ and a temperature cooling factor of $t_f = 0.998$ every $e_t = 200$ epochs.

VI. Negative Reinforcement

When we run a HORN using the backtrack-points method of the previous section and we obtain a feasible locally-optimal solution, we often would like to be able to run the network further and perhaps obtain a better local minimum. We now propose a method for achieving this. The idea of this method is to apply an algorithm at the end of the network simulation to reduce the attractiveness of (i.e. negatively reinforce) the current local minimum in order to encourage the network to search for other solutions. To do so, we alter the network topology according to the following algorithm:

Algorithm 2

1) Construct the set $\mathcal{N}$ of all neurons x such that
   a) $V_x > 0.5$.
   b) x corresonds to a hypothesis $h \in \mathcal{H}^A$.
2) Save the current network state.
3) Add to the network a hyperedge of degree $|\mathcal{N}|$ whose members are the elements of $\mathcal{N}$ and with weight $-\Gamma_3 c_\infty$, where $\Gamma_3 > 0$.
4) Restart the network simulation from the current state. [1]
5) After the simulation, check the solution corresponding to the network state:
   a) If the solution is invalid, restore the state saved in step 2 and exit.
   b) Else, reapply the algorithm.

Fig. 2 shows the application of this technique to `raa180`, after the main simulation had ended and the network settled in a local minimum at a solution of cost 24,713. The algorithm ran for a total of 9 iterations, using $\Gamma_3 = 20$ and a high temperature decrease factor of $t_f = 0.83$ every $e_t = 200$ epochs during this phase of the simulation. The first 6 iterations all resulted in solution improvements. The seventh iteration failed to move the network from its current local minimum, and it took one more iteration to move it to a better solution. Finally, the ninth iteration resulted in an invalid solution (not shown in the figure), so the network was restored to the last good state giving a solution of cost 21,092.

[1]For best results, it is recommended that the network simulation restarts at a high temperature (like the original initial temperature) but with a much higher cooling rate than the original

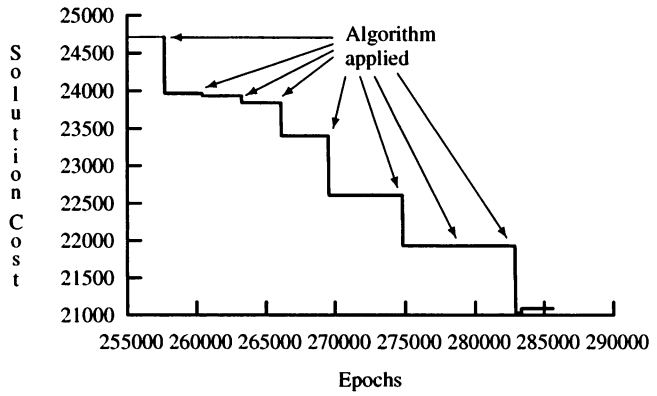

Fig. 2. Application of negative reinforcement to `raa180` after the main simulation had ended.

The epoch and CPU-time breakdown is shown in Table I. We can see that the negative reinforcement technique resulted in a cost reduction of 14.7% at the expense of increasing the number of epochs by 10.7%—which is only a 2.8% increase in execution time because, as discussed in the next section, epochs close to the end of the simulation take less time than epochs at the beginning.

VII. Implementation & Parallelization Issues

A. Hyperedge Computation

Due to the symmetry of the HORN employed, we find that an efficient way of simulating the network in each epoch is to process the *hyperedges* one at a time, not the neurons. This means that for each hyperedge $T^{(k)}_{i_1 \ldots i_k}$, we do the multiplication just once for all the neurons $i_1 \ldots i_k$ in the hyperedge to get the product P_T, and then update the state of each neuron for this edge by $\Delta u_i = P_T / V_i$. After all the hyperedges are "fired," each neuron's output is updated by equation (4). Thus, instead of doing $k(k-1)$ multiplications per edge, we do $2k$ multiplications per edge. This gives a good speed advantage in case of large hyperedges, which happens frequently in this kind of problem (e.g. for `raa180`, $k_{\text{avg}} = 25.9$).

As the network simulation advances, more and more hyperedges become *inactive*. This happens when two or more of the neurons participating in the hyperedge have a 0 output. Thus,

TABLE I
Epoch and CPU Time Breakdown for a Typical Run

milestone	epoch number	CPU time
network generation	—	7.9 min.
time to first backtrack point	800	9.4 min.
finding first solution	257,600	53.3 min.
end of negative reinforcement phase	285,200	54.8 min.

to speed up the simulation in this phase, we flag any such hyperedge and store with it which neurons are responsible for its deactivation, and then in any subsequent epoch we merely check those neurons, and if we find they are still inactive we can skip this edge. This technique greatly speeds up the simulation especially in its later phases.

B. Parallelization

One of the main attractions of connectionist methods is their natural capacity for massive parallelism. We expect that if the above technique for evaluating each hyperedge once is carried over into parallel techniques, then the most natural division of labour would be that each processing unit be responsible for a subset of the hyperedges. Another (much smaller) group of processors would be responsible for updating the V_i's according to equation (4).

For maximum performance, there is no need for tight synchronization between the group of processors responsible for firing the hyperedges and those responsible for firing the neurons. If it happens that the product of a hyperedge is computed using some V_i values from the current epoch and some from the previous epoch, this will only increase the level of randomness in the system and could contribute to escaping local minima.

VIII. Conclusions and Future Directions

In this paper, we presented two techniques for improving the solution quality of the HORN method of cost-based abduction. The backtrack-points technique (Sect. V) allowed feasible solutions, of reasonable quality, to be returned for `raa180`, where the vanilla approach failed repeatedly to return feasible solutions for this problem. The negative reinforcement technique (Sect. VI) significantly improved solution cost while increasing run-time almost negligibly.

The instance `raa180` is a particularly-difficult (see Sect. III.C) instance of CBA. Comparing HORN's to ILP, we find that HORN's, thus far, are not able to find the optimal solution for this instance (although, of course, in previous work [5], HORN's found the optimal solution for a number of (easier) instances). We would hope that future research will lead to one of two results:

1) Techniques such as those presented in this paper will allow better solutions to be found by HORN's for instances in the class of `raa180`.
2) HORN's will be able to return good-quality solutions to CBA instances that are of a harder class of difficulty than `raa180`. ILP, ultimately, has worst-case exponential complexity. The run-time of ILP on `raa180` is more than 24 CPU-hours. As problem difficulty increases, there will come a point where ILP becomes impractical. The challenge is for HORN performance to continue to scale well as network size increases.

References

[1] A.M. Abdelbar, "An Algorithm for Finding MAP Explanations through Cost-Based Abduction," *Artificial Intelligence*, **104** (1998) 331-338.

[2] A.M. Abdelbar, "Designing high order recurrent networks for Bayesian belief revision," in: L. Medsker, and L.C. Jain, ed., *Recurrent Neural Networks: Design and Applications*, CRC Press, 1999, pp. 77-98.

[3] Ashraf M. Abdelbar, "Approximating Cost-Based Abduction Is NP-Hard," *Artificial Intelligence*, Vol. 159, No.1-2, Nov. 2004, pp. 231-239.

[4] A.M. Abdelbar, and H. Amer, "Applying Guided Evolutionary Simulated Annealing to Cost-Based Abduction," *Proceedings IEEE International Joint Conference on Neural Networks*, 2003, Vol. 3, pp. 2428-2431.

[5] A.M. Abdelbar, E.A.M. Andrews, and D.C. Wunsch, "Abductive Reasoning with Recurrent Networks," *Neural Networks*, Vol. 16, No. 5-6, 2003, pp. 665-673.

[6] A.M. Abdelbar, and M. Mokhtar, "A k-Elitist MAX-MIN Ant System Approach to Cost-Based Abduction," *Proceedings IEEE Congress on Evolutionary Computation*, 2003, Vol. 4, pp. 2635-2641.

[7] A.M. Abdelbar, and M. Hefny, "An Efficient LP-Based Admissible Heuristic for Cost-Based Abduction," *Journal of Experimental and Theoretical Artificial Intelligence*, 2005, *to appear*.

[8] E. Charniak, and S. Husain, "A New Admissible Heuristic for Minimal-Cost Proofs," *Proceedings AAAI National Conference on Artificial Intelligence*, 1991, pp. 446-451.

[9] E. Charniak, and S.E. Shimony, "Probabilistic Semantics for Cost-Based Abduction," *Proceedings AAAI National Conference on Artificial Intelligence*, 1990, pp. 106-110.

[10] E. Charniak, and S.E. Shimony "Cost-Based Abduction and MAP Explanation," *Artificial Intelligence*, Vol. 66, 1994, pp. 345-374.

[11] Y. Den, "Generalized Chart Algorithm: An Efficient Procedure for Cost-Based Abduction," *Proceedings Meeting of the Association for Computational Linguistics*, 1994, pp. 218-224.

[12] G.E. Hinton, and T.J. Sejnowski, "Learning and re-learning in Boltzmann machines," in: J.L. McClelland, D.E. Rumelhart, eds., *Parallel Distributed Processing* I, MIT Press, pp. 282-317, 1986.

[13] J.R. Hobbs, M.E. Stickel, D.E. Appelt, and P. Martin, "Interpretation as Abduction," *Artificial Intelligence*, Vol. 63, 1993, pp. 69-142.

[14] J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceedings National Academy of Science*, Vol. 79, 1982, pp. 2554-2558.

[15] M. Ishizuka, and Y. Matsuo, "SL Method for Computing a Near-Optimal Solution Using Linear and Non-linear Programming in Cost-Based Hypothetical Reasoning," *Proceedings Pacific Rim International Conference on Artificial Intelligence*, 1998, pp. 611-625.

[16] S. Kato, S. Oono, H. Seki, and H. Itoh, "Cost-Based Abduction Using Binary Decision Diagrams," *Proceedings Industrial and Engineering Applications of Artificial Intelligence*, 1999, pp. 215-225.

[17] S. Kato, H. Seki, and H. Itoh, "Cost-Based Horn Abduction and Its Optimal Search," *Proceedings Third International Conference on Automation, Robotics and Computer Vision*, 1994, pp. 831-835.

[18] S. Kato, H. Seki, and H. Itoh, "Parallel Cost-Based Abductive Reasoning for Distributed Memory Systems," *Proceedings Pacific Rim International Conference on Artificial Intelligence*, 1996, pp. 300-311.

[19] Y. Matsuo, and M. Ishizuka, "Two Transformations of Clauses into Constraints and Their Properties for Cost-Based Hypothetical Reasoning," *Proceedings Pacific Rim Conference on Artificial Intelligence*, 2002, pp. 118-127.

[20] Y. Ohsawa, and M. Ishizuka, "Networked Bubble Propagation: A Polynomial-Time Hypothetical Reasoning Method for Computing Near-Optimal Solutions," *Artificial Intelligence*, Vol. 91, 1997, pp. 131-154.

[21] Y. Ohsawa, and M. Yachida, "An Extended Polynomial Solvability of Cost-Based Abduction," *Proceedings International Joint Conference on Artificial Intelligence (poster session abstracts)*, 1997, p. 79.

[22] C. Peterson, and E. Hartman, "Explorations of the Mean Field Theory Learning Algorithm," *Neural Networks*, Vol. 2, pp. 475-494, 1989.

[23] G. Pinkas, "Reasoning, Nonmonotonicity and Learning in Connectionist Networks that Capture Propositional Knowledge," *Artificial Intelligence* **77** (1995) 203-247.

[24] E. Santos Jr., "On the Generation of Alternative Explanations with Implications for Belief Revision," *Proceedings Seventh Conference on Uncertainty in AI*, 1991, pp. 339-347.

[25] E. Santos Jr., "A Linear Constraint Satisfaction Approach to Cost-Based Abduction," *Artificial Intelligence*, Vol. 65, 1994, pp. 1-27.

[26] E. Santos, Jr., and E.S. Santos, "Polynomial Solvability of Cost-Based Abduction," *Artificial Intelligence*, Vol. 86, 1996, pp. 157-170.

[27] T.J. Sejnowski, "Higher-order Boltzmann machines," In: J. Denker, ed., *Neural Networks for Computing*, American Institute of Physics, New York, 1986, pp. 398-403.